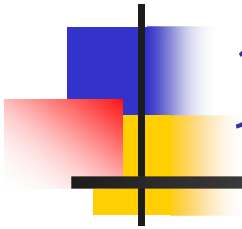
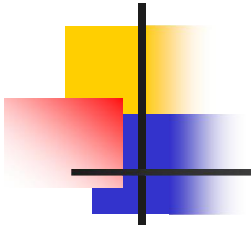
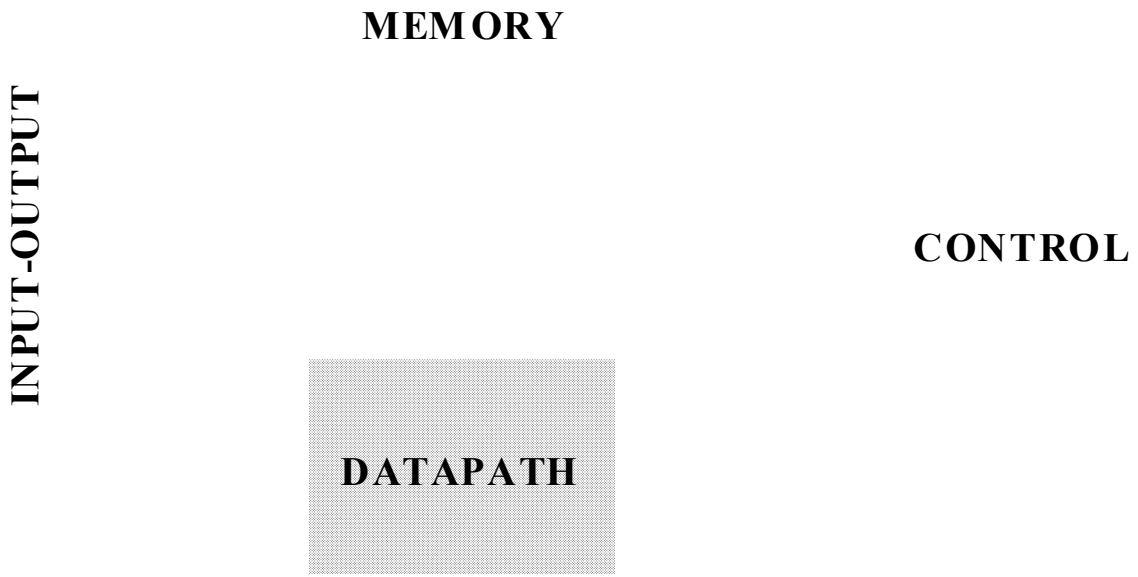


第五章 数字集成电路基本模块

5.2 加法器

计算机组成

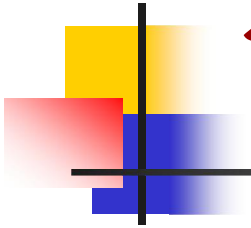




数据通路（运算器）

- 加法器、移位器、乘法器等组成，加法器是运算器的核心电路

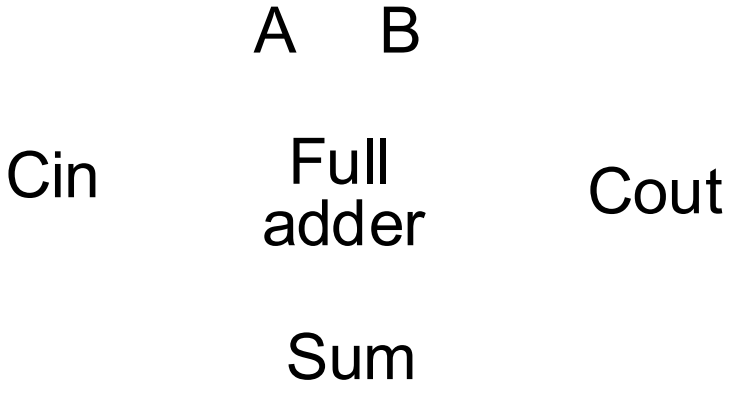
加法器



- 二进制加法
- 加法器结构设计
- 加法器电路设计

二进制加法

- 两个n位二进制数据与进位输入信号进行加法运算，产生1个n位的加法和与一个进位输出信号



$$S = A \oplus B \oplus C_i$$

$$= ABC_i + ABC_i + ABC_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

全加器：1位加法器

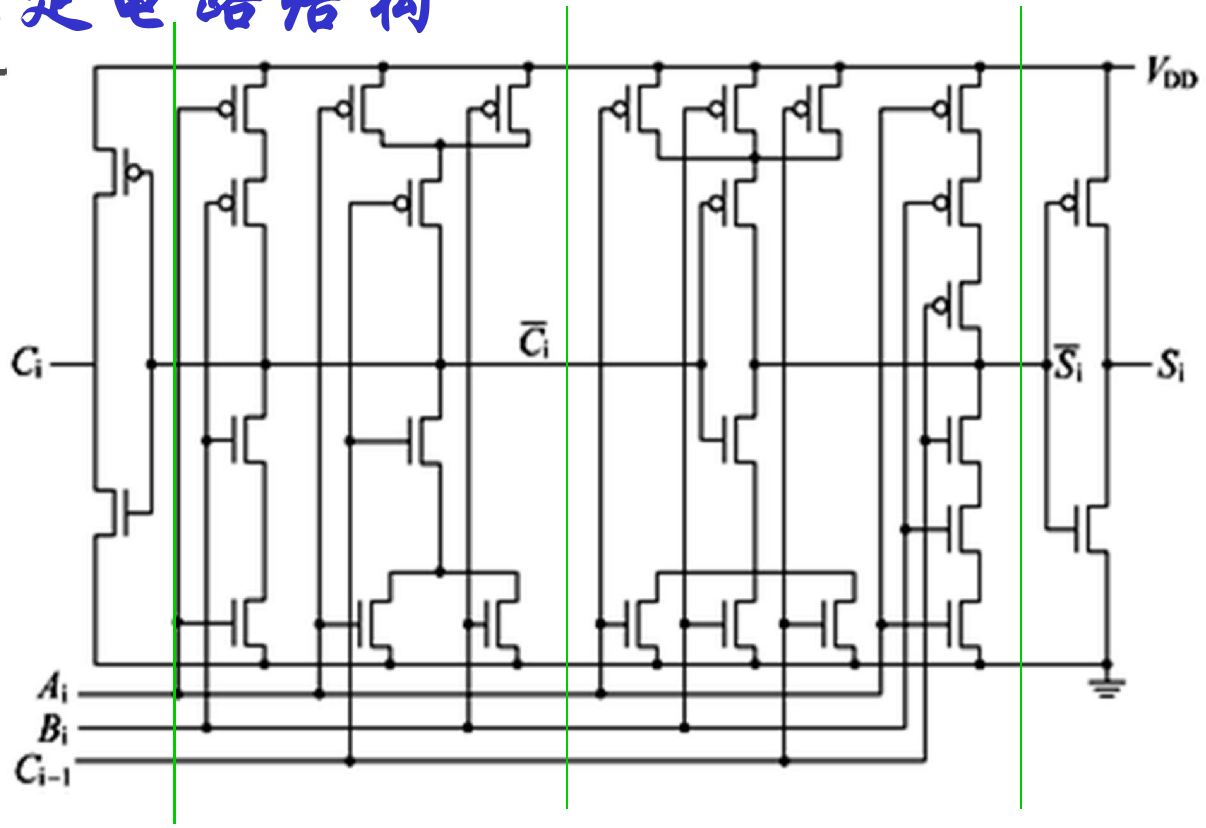
$$S = \overline{A}\,\overline{B}\,C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$
$$= \left(\overline{A}B + A\overline{B}\right)\overline{C} + \left(AB + \overline{A}\,\overline{B}\right)C$$
$$= \left[\left(\overline{A}B + A\overline{B}\right) + C\right]\cdot\left[\left(AB + \overline{A}\,\overline{B}\right) + \overline{C}\right]$$
$$= \overline{\left(AB + \overline{A}\,\overline{B}\right)\cdot\overline{C} + \left(\overline{A}B + A\overline{B}\right)\cdot C}$$
$$= \overline{\left(AB + \overline{A}\,\overline{B} + C\right)\cdot\overline{C} + \left(\overline{A}B + A\overline{B}\right)\cdot\left(AB + \overline{A}\,\overline{B} + C\right)}$$
$$= \overline{\overline{AB + \overline{A}\,\overline{B} + C}\;\overline{C}} + \overline{\left(AB + \overline{A}\,\overline{B}\right)\overline{AB + \overline{A}\,\overline{B} + C}}$$

A	B	C	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$CO = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$
$$= AB + AC + BC = AB(A + B) + (A + B)C$$
$$= (A + B)\cdot(AB + \overline{A}\,\overline{B} + C)$$
$$= \overline{\overline{A}\,\overline{B} + \overline{AB + \overline{A}\,\overline{B} + C}}$$

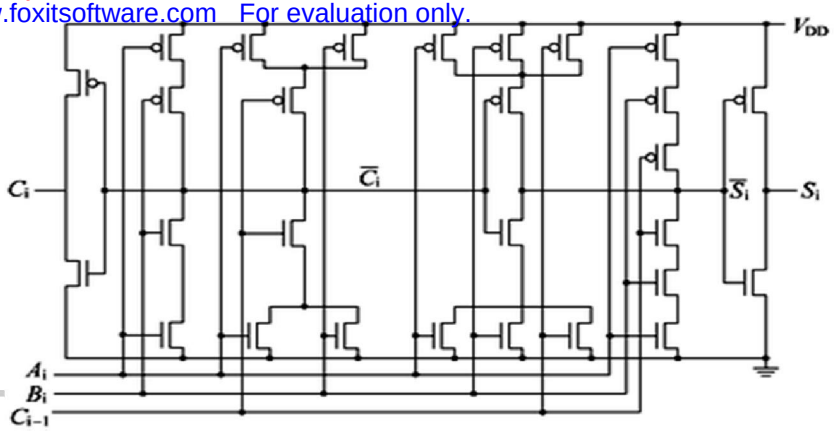
一位全加器实现方案

确定电路结构



经过变换使串联管子的数目减少，2个AOI门刚好形成镜像对称的电路形式

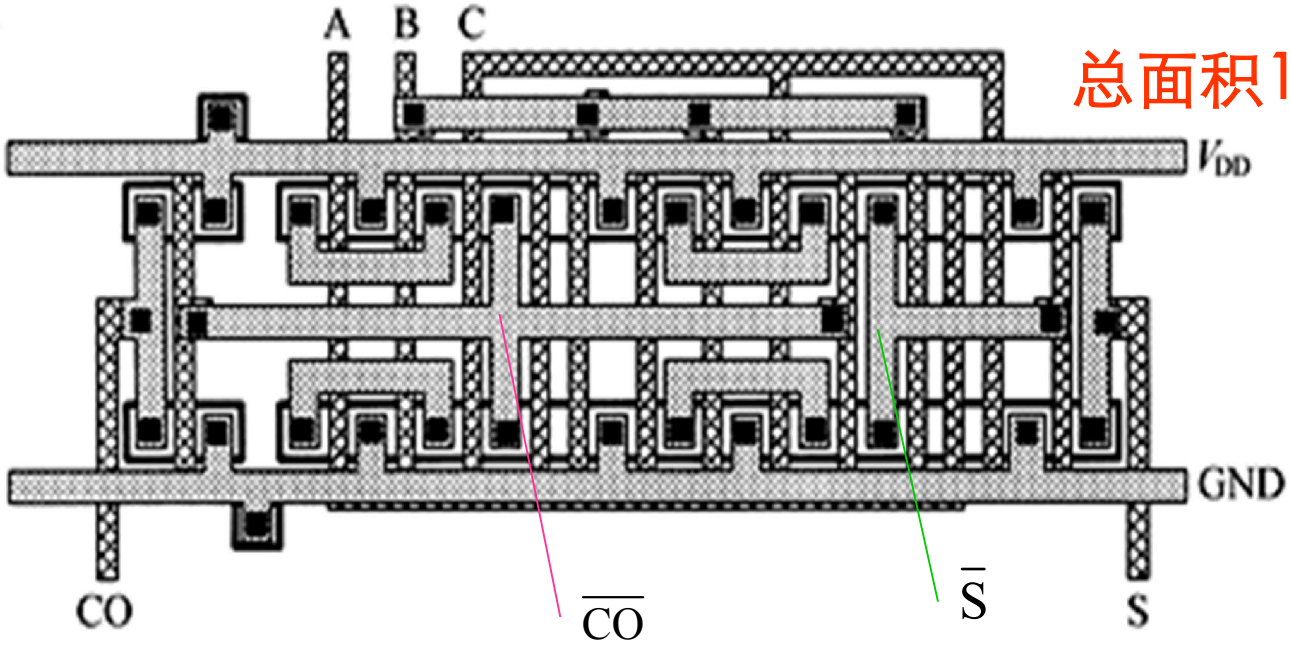
全加器版图设计



1.减小面积：把所有MOS管都取相同的W/L，尽可能小尺寸。

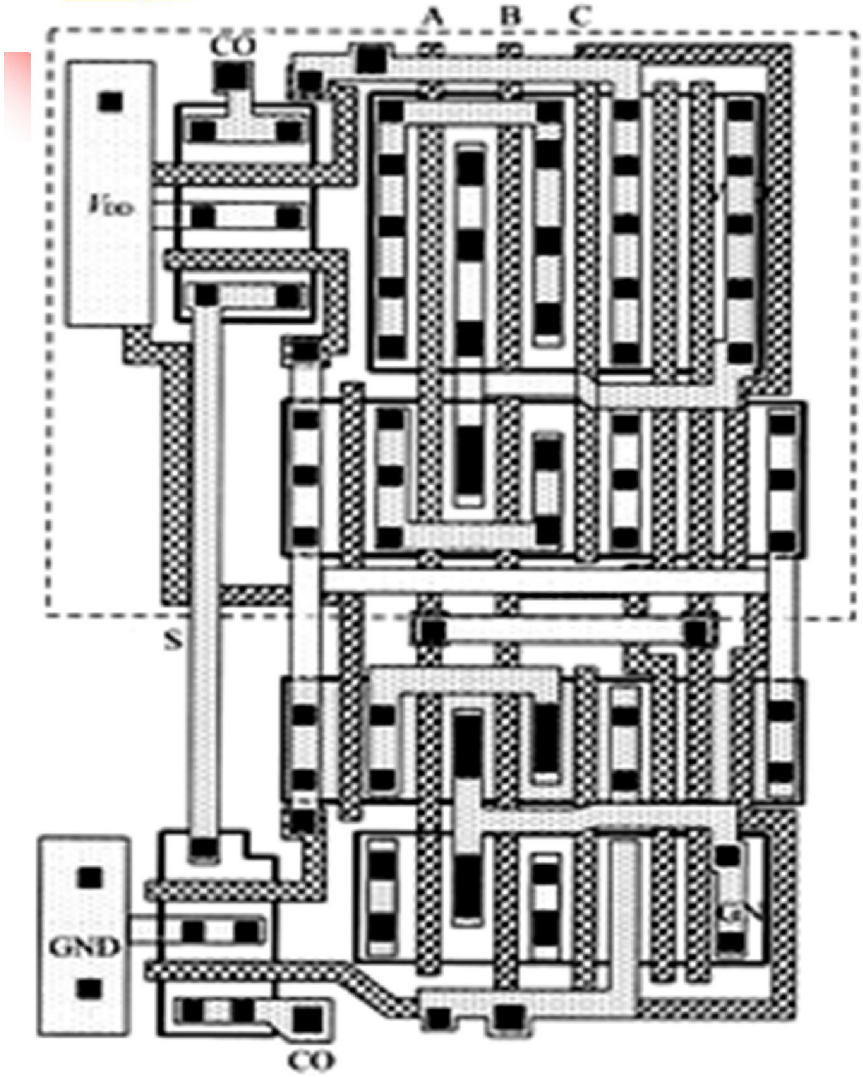
2.简化版图设计：所有多晶硅栅都是简单的直条矩形，并按同一方向排列。

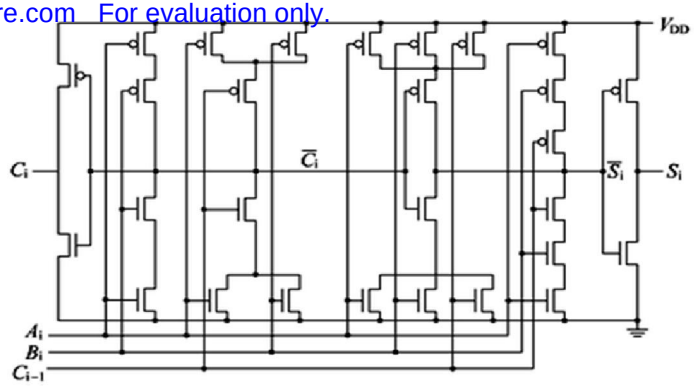
3.便于做阱区：所有PMOS管集中在上半部，所有NMOS管集中在下半



总面积1134 μm^2

全加器版图优化

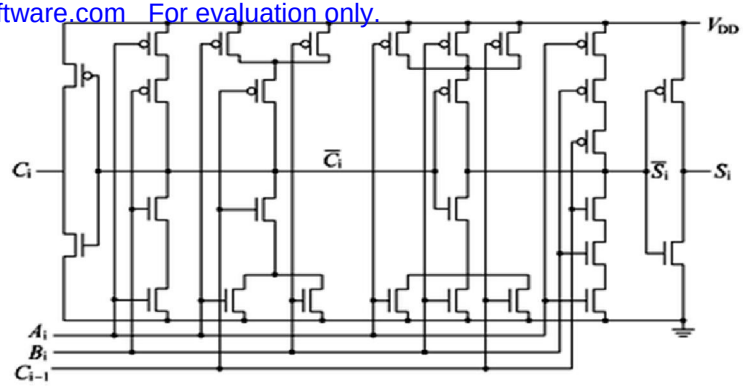




- 1.考虑到空穴迁移率比电子迁移率低，PMOS宽度加大。
- 2.关键路径上的延迟：产生CO的AOI门中所有MOS管的尺寸要大一些。
- 3.电容的影响：在产生S的AOI门中为避免3个串联MOS管中间结点电容的影响，把晚来的信号C接到最靠近输出结点的MOS管。

与未优化前相比，面积增加了17%，延迟减少了50%。

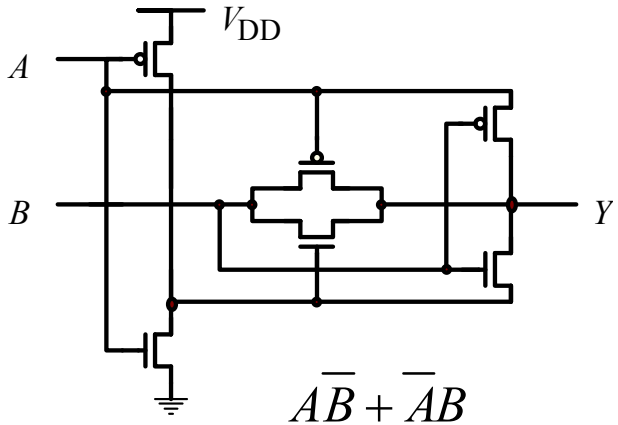
用传输门实现全加器

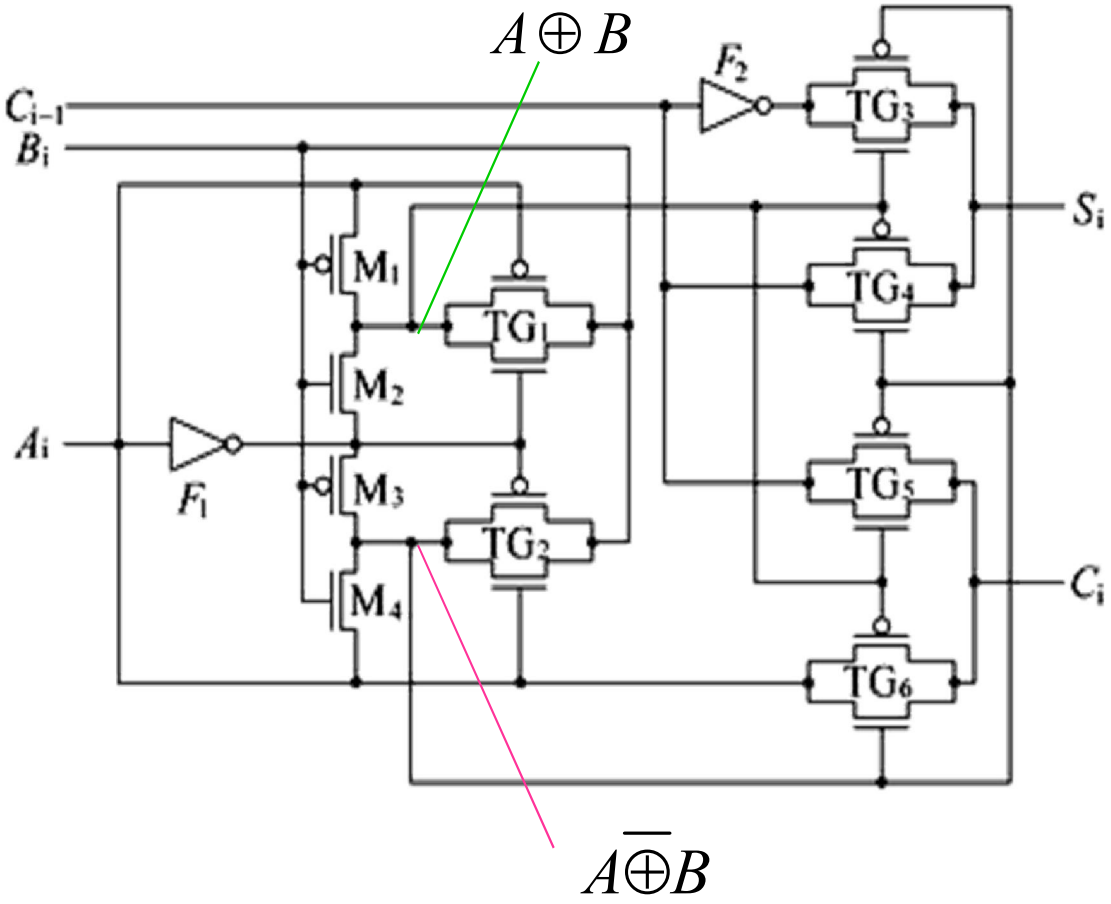


只有20个MOSFET构成！

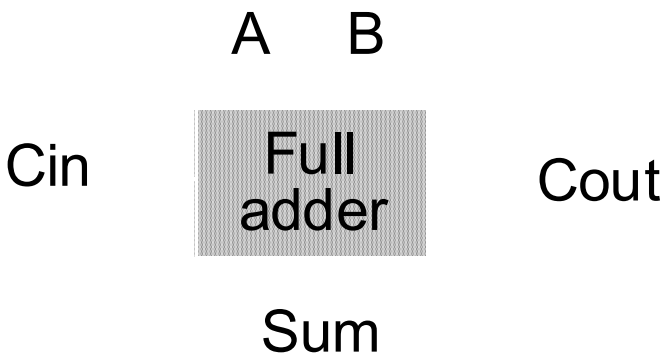
$$S = (A\overline{\oplus}B)C + (A \oplus B)\overline{C}$$

$$CO = (A \oplus B)C + (A\overline{\oplus}B)A$$





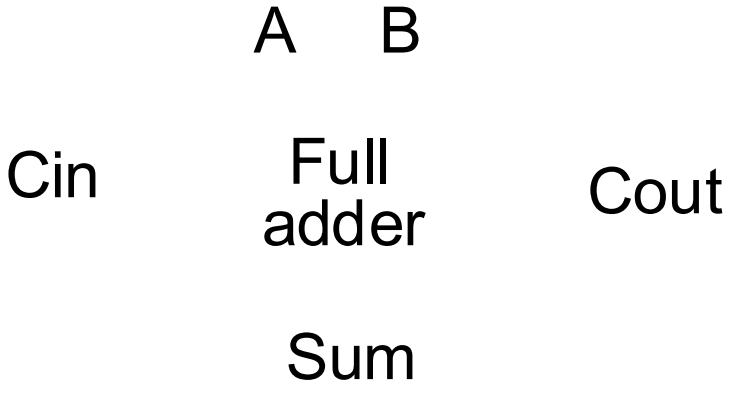
Full-Adder



A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

多位二进制加法器

- 两个n位二进制数据与进位输入信号进行加法运算，产生1个n位的加法和与一个进位输出信号



$$S = A \oplus B \oplus C_i$$

$$= ABC_i + ABC_i + ABC_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

Express Sum and Carry as a function of P, G, D

- Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Delete = $\overline{A}\,\overline{B}$

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

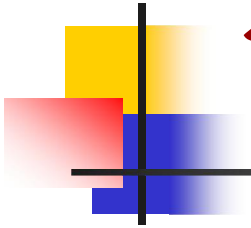
A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Can also derive expressions for S and C_o based on D and P

Note that we will be sometimes using an alternate definition for

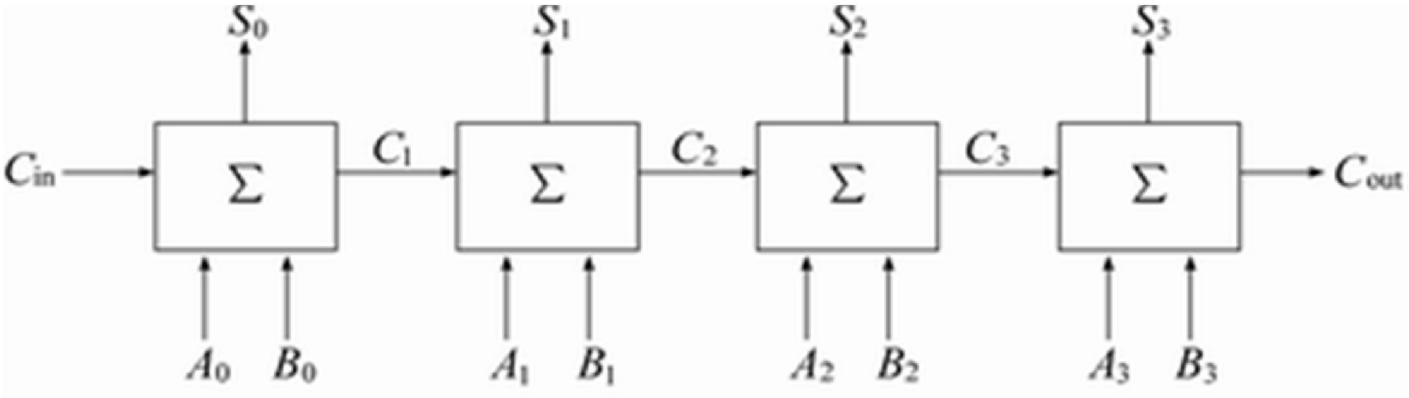
Propagate (P) = A + B

加法器



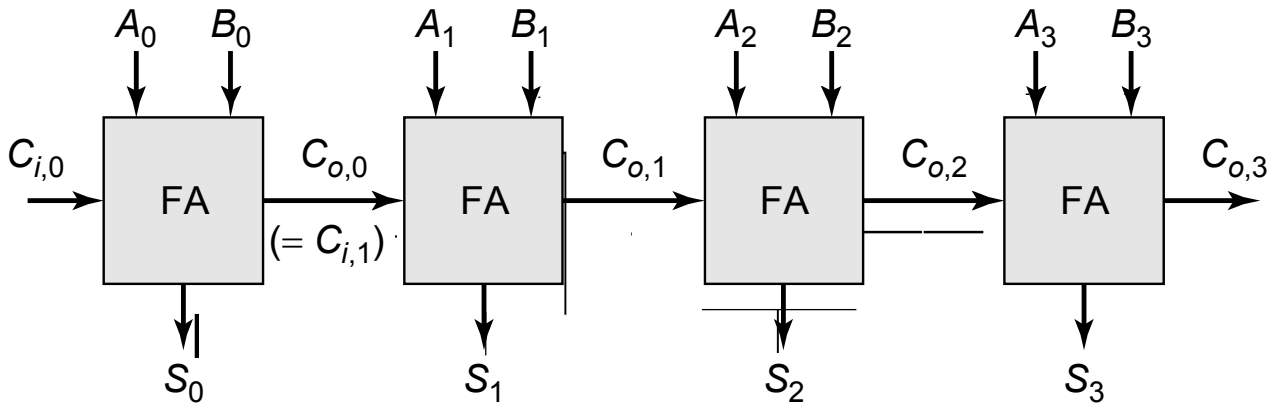
- 二进制加法
- 加法器结构设计
- 加法器电路设计

行波进位加法器（RCA）



- 行波进位的*并行加法器*
- 问题：高位要等待低位产生进位

The Ripple-Carry Adder



Worst case delay linear with the number of bits

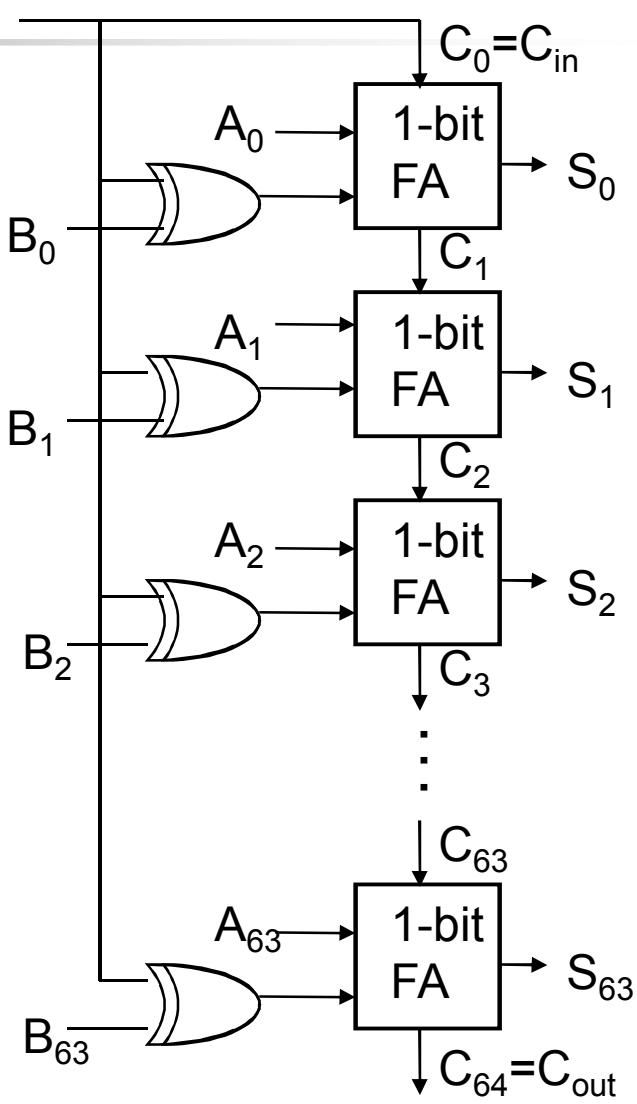
$$t_d = \mathrm{O}(N)$$

$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$$

Goal: Make the fastest possible carry path circuit

A 64-bit Adder/Subtractor

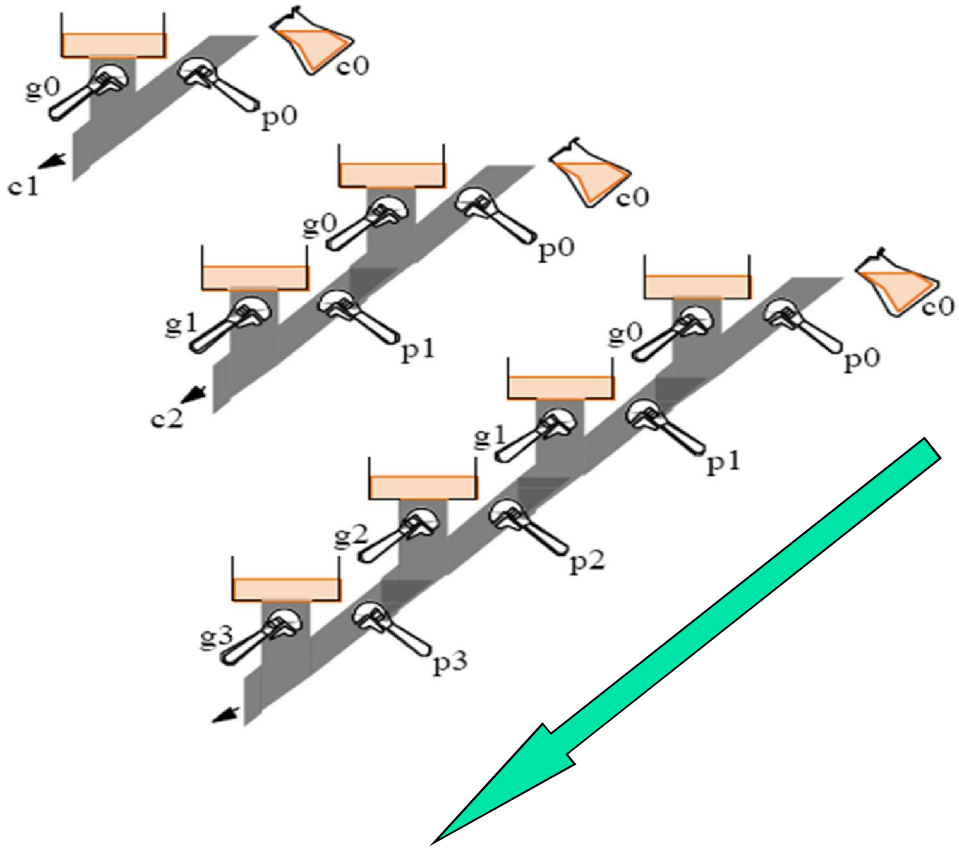
- 利用64个全加器实现的行波进位加法器 (RCA)
- 减法功能 – 根据控制信号（add/subt）对被操作数进行取反和加一操作，实现二进制减法
- RCA
  - 优点: simple logic, so small (low cost)
  - 缺点: slow (O(N) for N bits) and lots of glitching (so lots of energy consumption)



加法器中的进位链

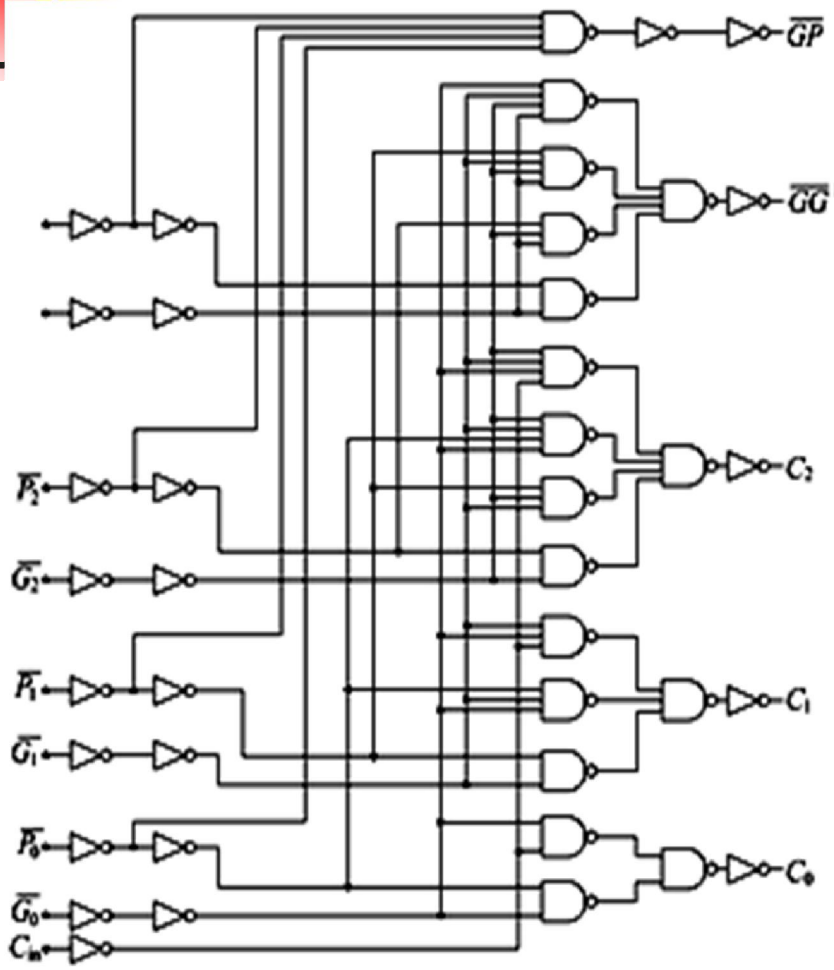
$$C_i = A_iB_i + (A_i \oplus B_i)C_{i-1} = G_i + p_iC_{i-1}$$

$G_i = A_iB_i$ 进位产生函数　　$P_i = A_i \oplus B_i$ 进位传递函数



当Gi=1时不管低位的进位是什么值，本位的进位输出都是“1”；当Pi=1时则把低位的进位输出Ci-1直接作为本位的进位输出传送下去。

加法器中的进位链



多位加法器：需要组进位产生函数GG和组进位传递函数GP，利用GG和GP以及最低位进位输入直接产生组进位输出。

$$C_0 = \overline{\overline{\left(\overline{G_0}\ \overline{P_0}\right)} \cdot \overline{\left(\overline{G_0}\ \overline{C_{in}}\right)}}$$

$$= \overline{\left(\overline{G_0}\ \overline{P_0}\right)} \cdot \overline{\left(\overline{G_0}\ \overline{C_{in}}\right)}$$

$$= \left(G_0 + P_0\right) \cdot \left(G_0 + C_{in}\right)$$

$$= G_0 + G_0 P_0 + P_0 C_{in}$$

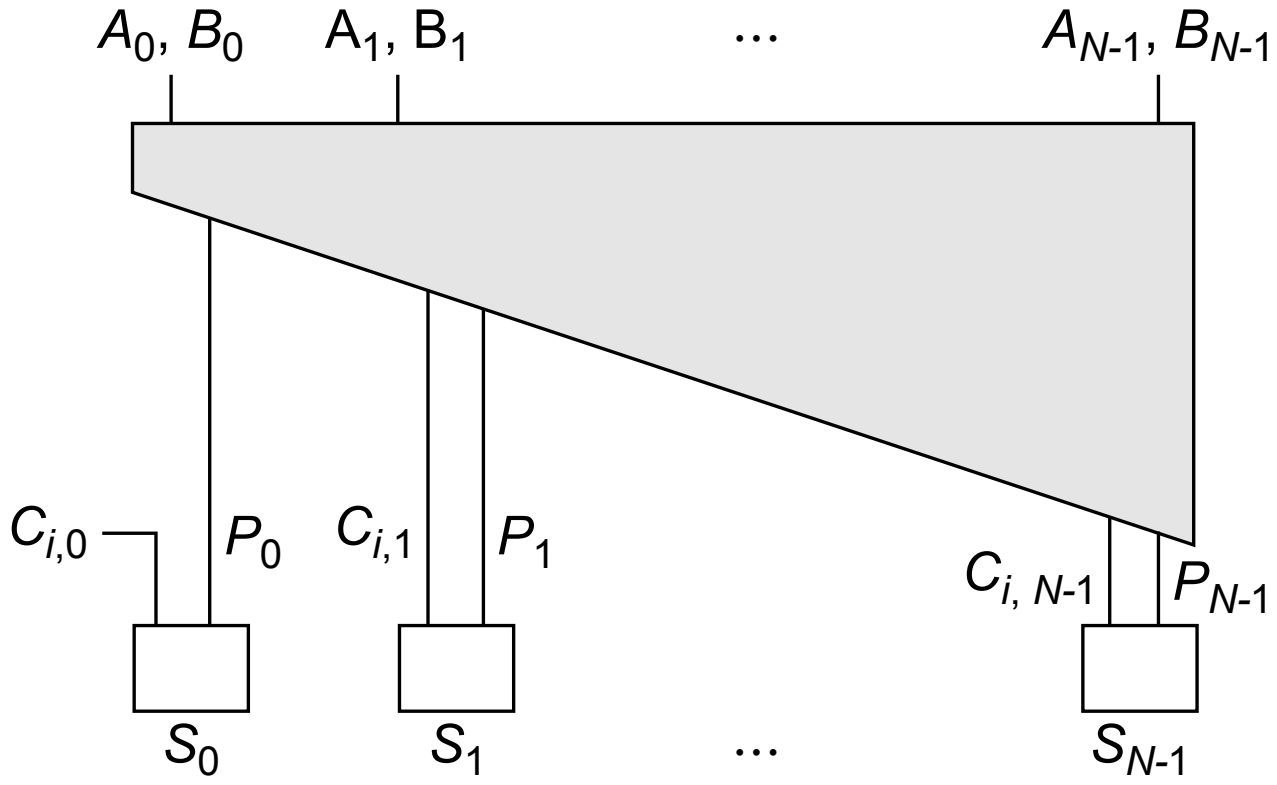
$$= G_0 + P_0 C_{in}$$

$$GG = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

$$GP = P_3 P_2 P_1 P_0$$

$$C_3 = GG + GPC_{\text{in}}$$

超前进位加法器（CLA）



$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

Carry-Lookahead Adder

$$c_i = \bar{a}_i b_i c_{i-1} + a_i \bar{b}_i c_{i-1} + a_i b_i = g_i + p_i c_{i-1}$$

$$c_{i+1} = g_{i+1} + p_{i+1} c_i$$
$$= g_{i+1} + p_{i+1}(g_i + p_i c_{i-1})$$
$$= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_{i-1}$$

$$c_{i+2} = g_{i+2} + p_{i+2} c_{i+1}$$
$$= g_{i+2} + p_{i+2}(g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_{i-1})$$
$$= g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_{i-1}$$

Carry-Lookahead Adder

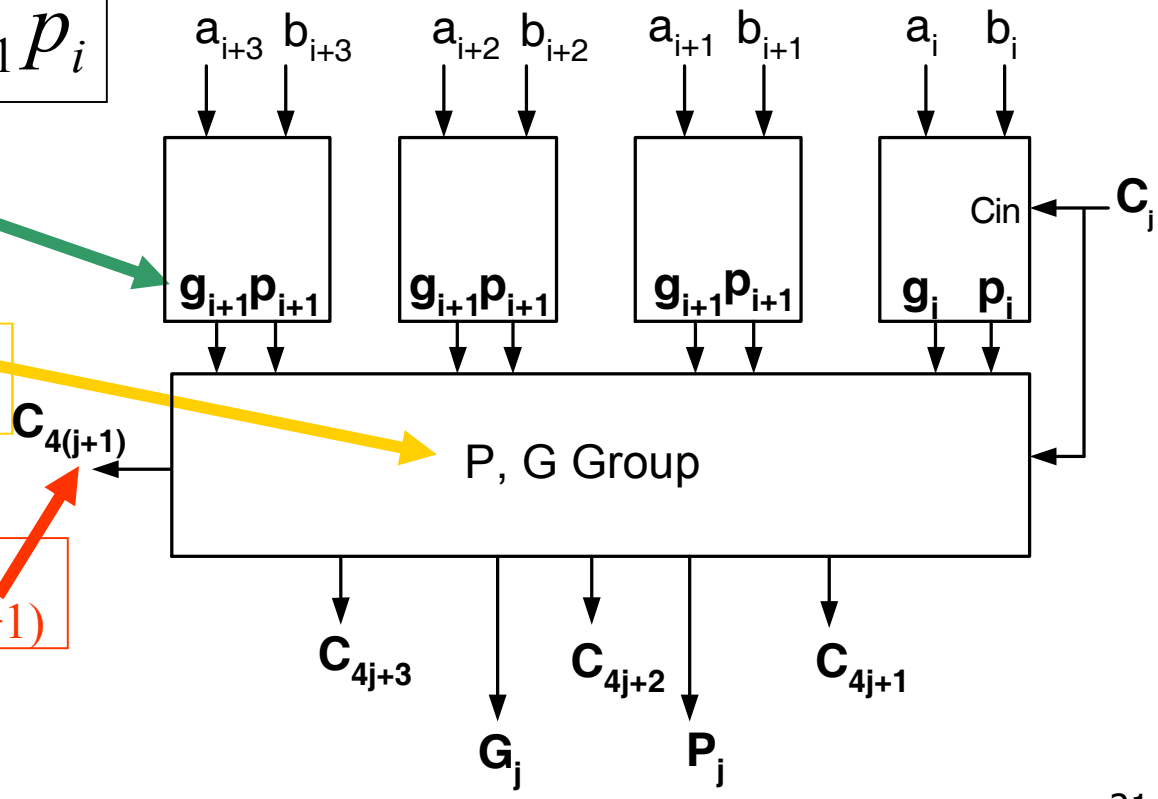
$$G_j = g_{i+3} + p_{i+3}g_{i+2} + p_{i+3}p_{i+2}g_{i+1} + p_{i+3}p_{i+2}p_{i+1}g_i$$

$$P_j = p_{i+3}p_{i+2}p_{i+1}p_i$$

计算p, g信号模块

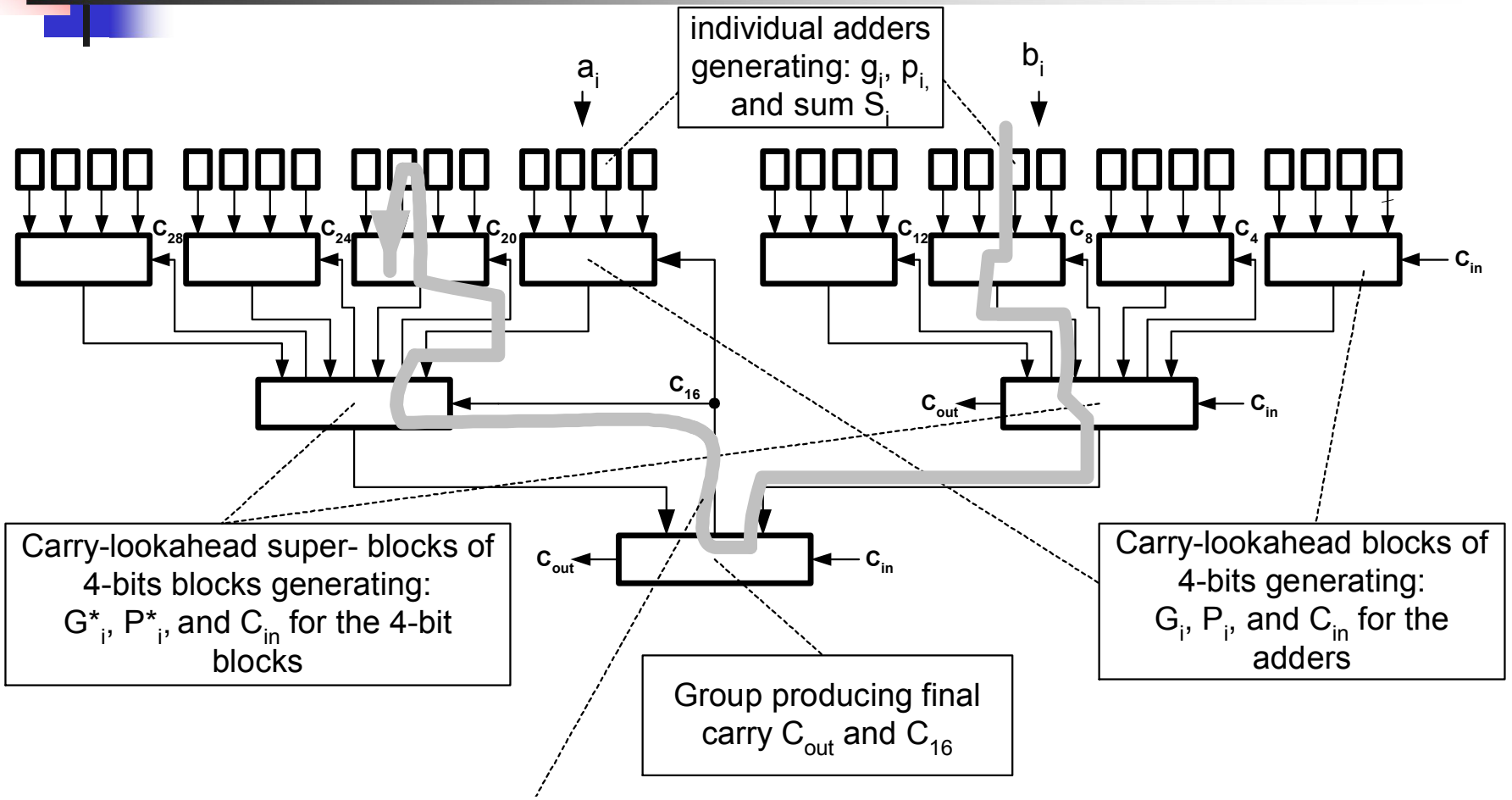
计算组进位P， G

输出组进位信号$C_{4(j+1)}$

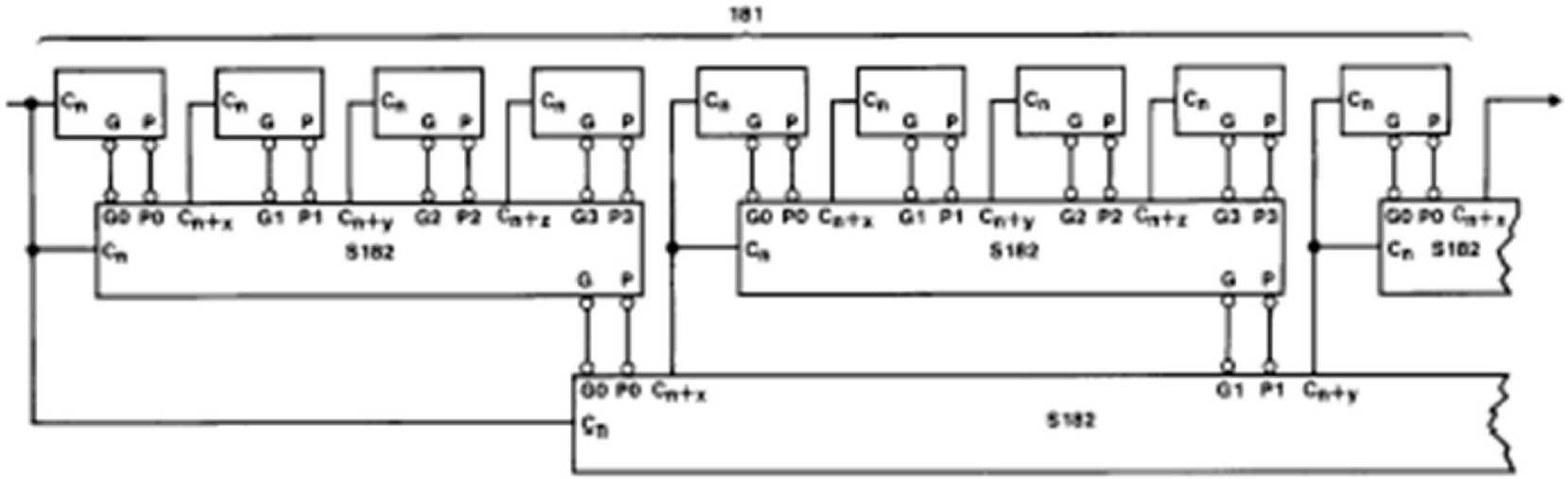


$$c_{4(j+1)} = G_i + P_i c_{4j}$$

32-bit Carry Lookahead Adder



配有进位链的加法器



- 多位加法器可以分成4位一组配进位链;
- 用第二层进位链产生组进位输出

超前进位：电路结构

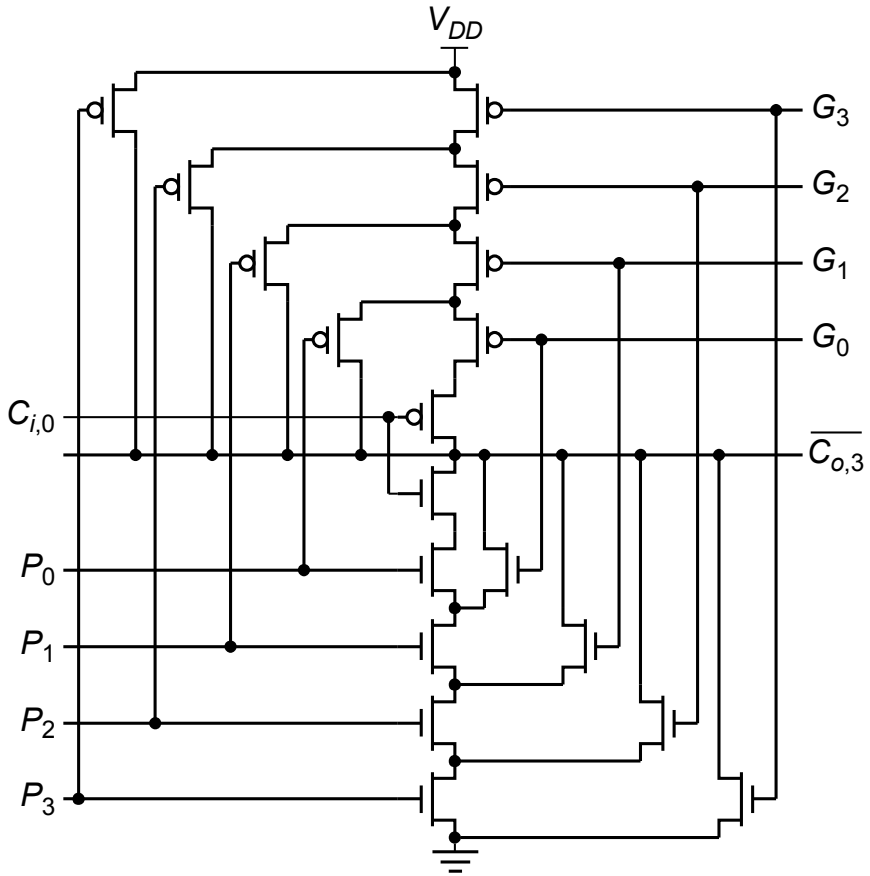
Expanding Lookahead equations:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

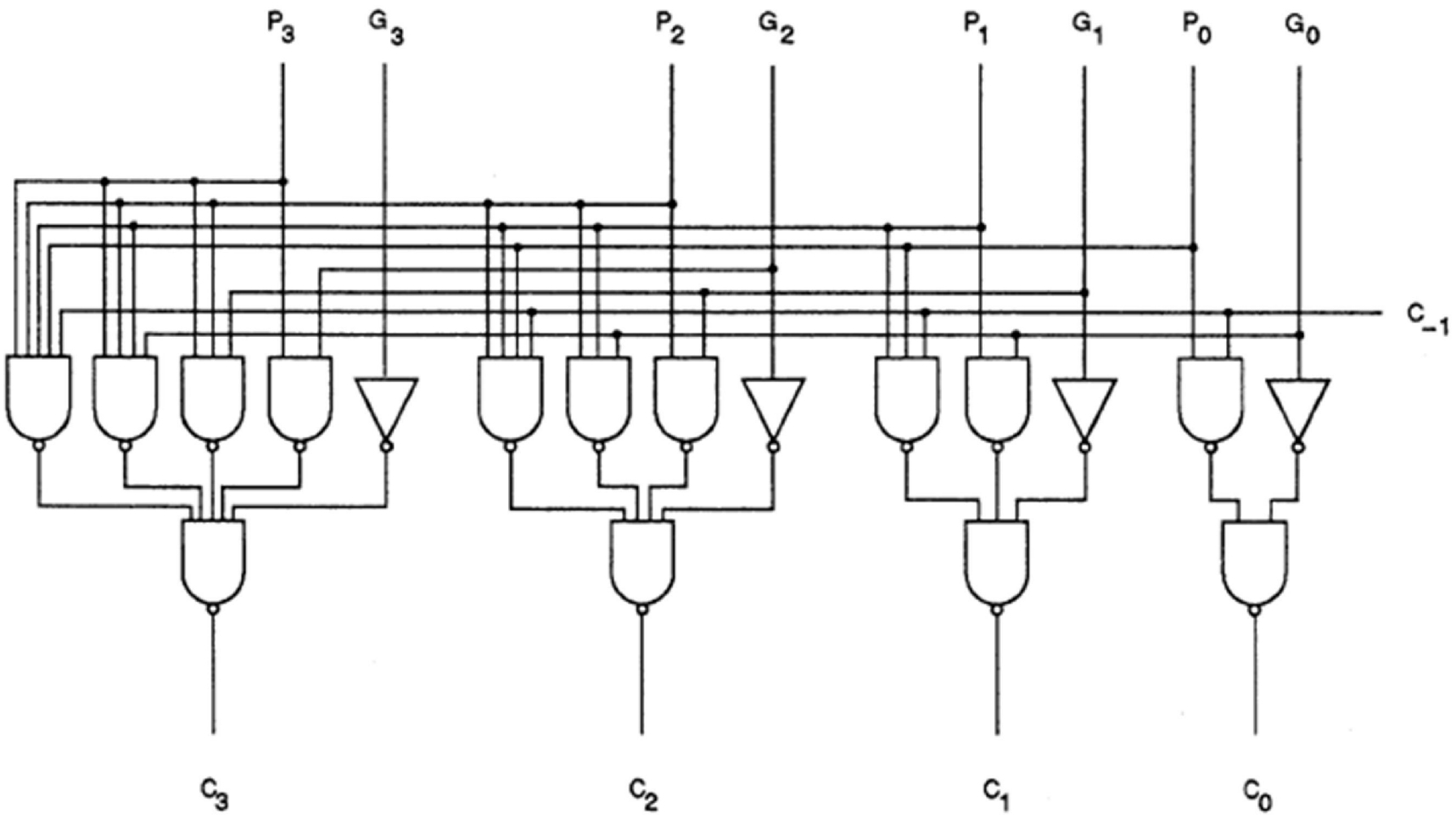
All the way:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0})))$$

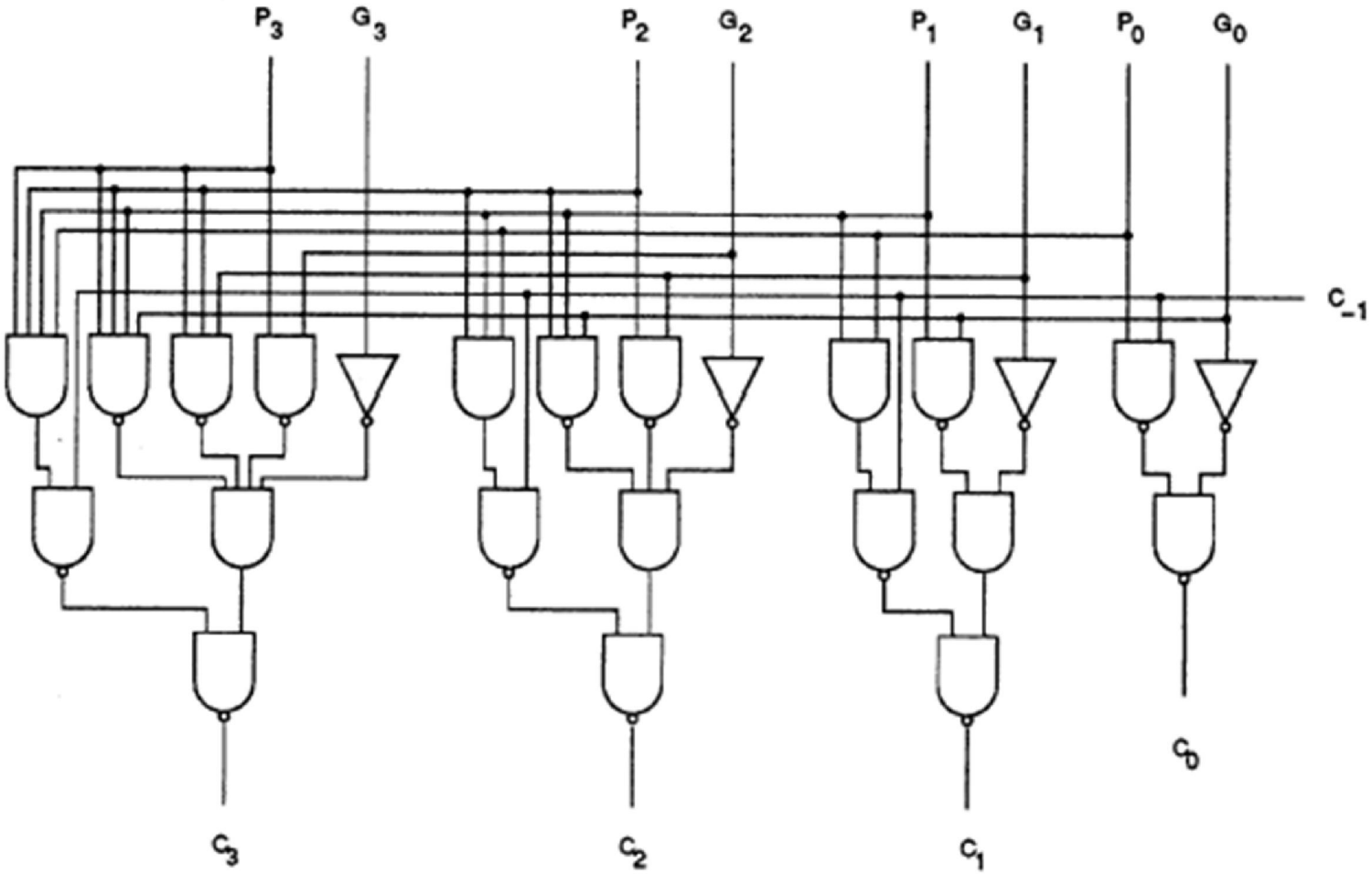
$$\begin{aligned} C_3 &= G_3 + P_3 C_2 \\ &= G_3 + P_3 G_2 + \\ & P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \\ & + P_3 P_2 P_1 P_0 C_{in} \end{aligned}$$



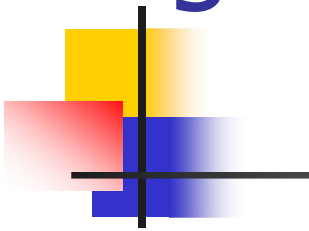
Two-Levels of Logic Implementation of the Carry Block

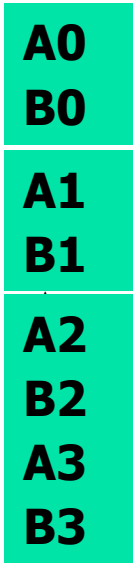


Three-Levels of Logic Implementation of the Carry Block (restricted fan-in)



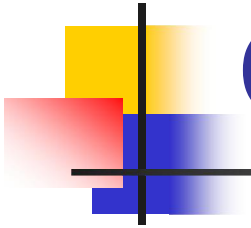
Logarithmic Look-Ahead Adder





F

$t_p \sim \log_2(N)$



Carry Lookahead Trees

$$C_{o,0} = G_0 + P_0 C_{i,0}$$

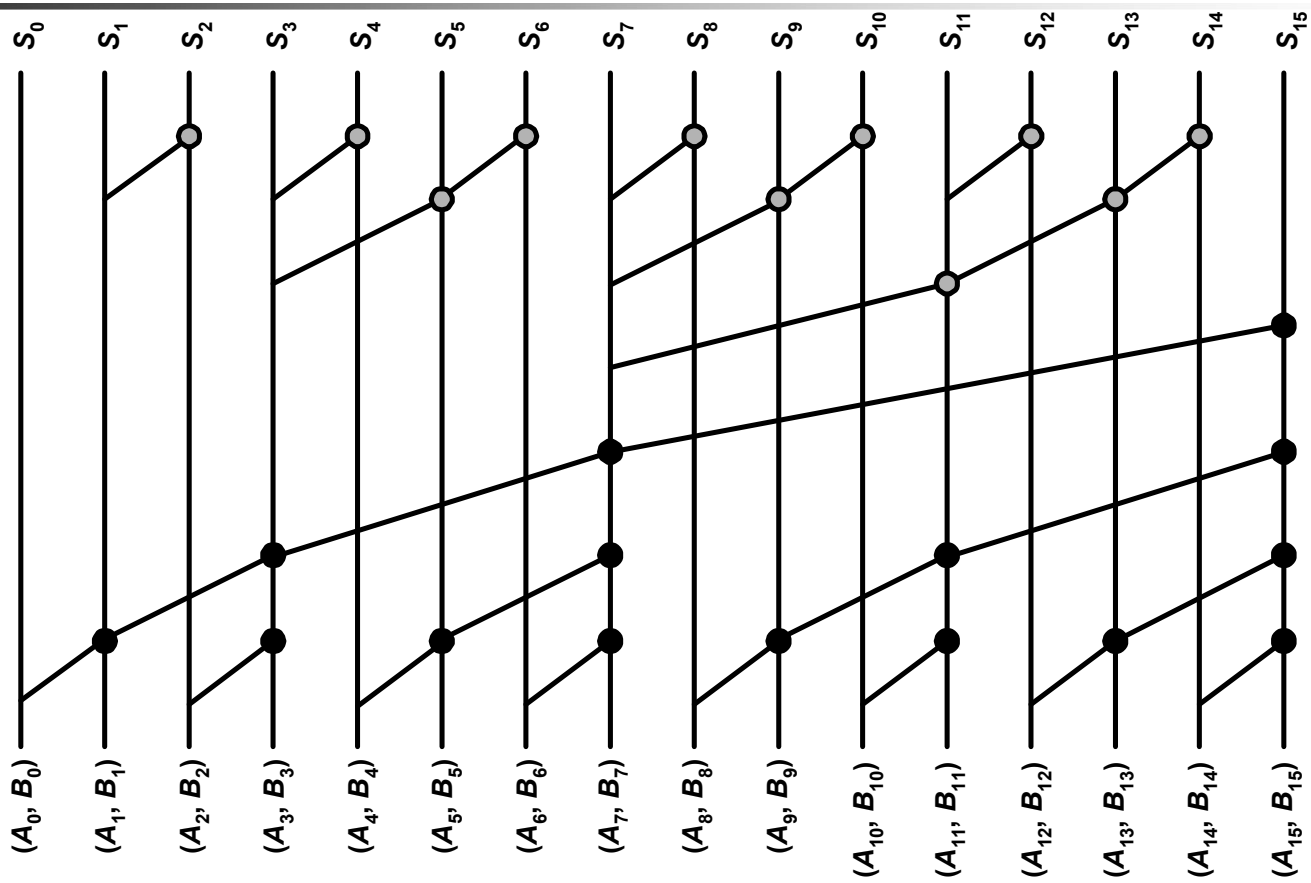
$$C_{o,1} = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$$

$$C_{o,2} = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$$

$$= (G_2 + P_2 G_1) + (P_2 P_1)(G_0 + P_0 C_{i,0}) = G_{2:1} + P_{2:1} C_{o,0}$$

Can continue building the tree hierarchically.

Tree Adders



Brent-Kung Tree

超前进位加法器的改进-ling算法

- **Ling**型加法器是一种改进的超前进位结构，考察四位一组的超前进位链：

$$c_i = g_i + p_i c_{i-1} = g_i + t_i c_{i-1} =$$

$$= g_i + t_i g_{i-1} + t_i t_{i-1} g_{i-2} + t_i t_{i-1} t_{i-2} g_{i-3} + t_i t_{i-1} t_{i-2} t_{i-3} c_{i-4}$$

- **Ling**算法中用伪进位信号**h**来代替进位信号

$$h_i = c_i + c_{i-1}$$

$$p_i h_i = p_i (c_i + c_{i-1}) = p_i (g_i + p_i c_{i-1} + c_{i-1}) = p_i g_i + p_i c_{i-1} = p_i c_{i-1}$$

$$g_i h_i = g_i (c_i + c_{i-1}) = g_i (g_i + p_i c_{i-1} + c_{i-1}) = g_i + g_i c_{i-1} = g_i$$

$$p_i h_i = p_i(c_i + c_{i-1}) = p_i(g_i + p_i c_{i-1} + c_{i-1}) = p_i g_i + p_i c_{i-1} = p_i c_{i-1}$$

$$g_i h_i = g_i(c_i + c_{i-1}) = g_i(g_i + p_i c_{i-1} + c_{i-1}) = g_i + g_i c_{i-1} = g_i$$

Ling加法器

把以上两式带入进位公式，得到伪进位信号和进位信号之间关系：

$$c_i = g_i + p_i c_{i-1} = g_i h_i + p_i h_i = t_i h_i$$

伪进位信号的传递过程：

$$h_i = c_i + c_{i-1} = g_i + p_i c_{i-1} + c_{i-1} = g_i + c_{i-1} = g_i + t_{i-1} h_{i-1}$$

因此同进位信号一样，伪进位信号可以逐位向上传递，用伪进位信号表示的**4**位一组的超前进位链逻辑为：

$$h_i = g_i + t_{i-1} h_{i-1} = g_i + t_{i-1}(g_{i-1} + t_{i-2} h_{i-2}) = g_i + g_{i-1} + t_{i-1} t_{i-2} h_{i-2} =$$

$$= g_i + g_{i-1} + t_{i-1} t_{i-2} g_{i-2} + t_{i-1} t_{i-2} t_{i-3} h_{i-3} =$$

$$= g_i + g_{i-1} + t_{i-1} g_{i-2} + t_{i-1} t_{i-2} g_{i-3} + t_{i-1} t_{i-2} t_{i-3} h_{i-4}$$

$$c_i = g_i + p_i c_{i-1} = g_i + t_i c_{i-1} =$$

$$= g_i + t_i g_{i-1} + t_i t_{i-1} g_{i-2} + t_i t_{i-1} t_{i-2} g_{i-3} + t_i t_{i-1} t_{i-2} t_{i-3} c_{i-4}$$

CLA-ling算法

$$h_i = g_i + t_{i-1} h_{i-1} = g_i + t_{i-1}(g_{i-1} + t_{i-2} h_{i-2}) = g_i + g_{i-1} + t_{i-1} t_{i-2} h_{i-2} =$$

$$= g_i + g_{i-1} + t_{i-1} t_{i-2} g_{i-2} + t_{i-1} t_{i-2} t_{i-3} h_{i-3} =$$

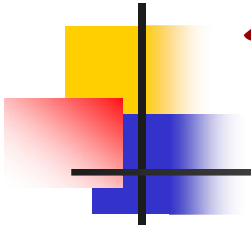
$$= g_i + g_{i-1} + t_{i-1} g_{i-2} + t_{i-1} t_{i-2} g_{i-3} + t_{i-1} t_{i-2} t_{i-3} h_{i-4}$$

同传统的**CLA**相比，**Ling**算法的逻辑级虽然没有减少，但进位链逻辑的扇入减少一个。在电路实现中，这意味着较短的路径延迟。伪进位的求和逻辑如下：

$$s_i = p_i \oplus c_{i-1} = p_i \oplus t_{i-1} h_{i-1}$$

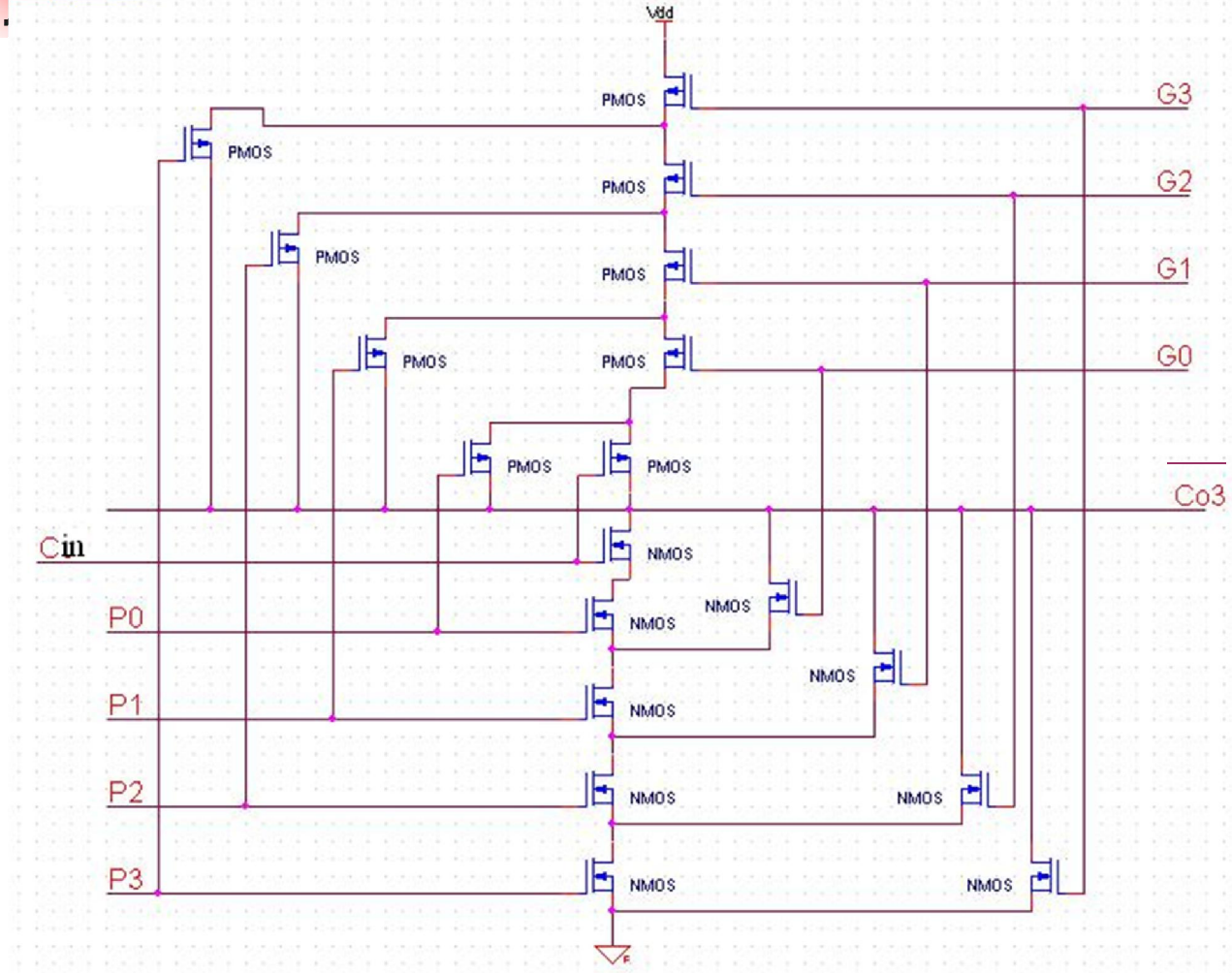
Ling算法是将关键路径——进位链上的逻辑转移到求和逻辑中，从而减少进位链的关键路径延迟。

加法器



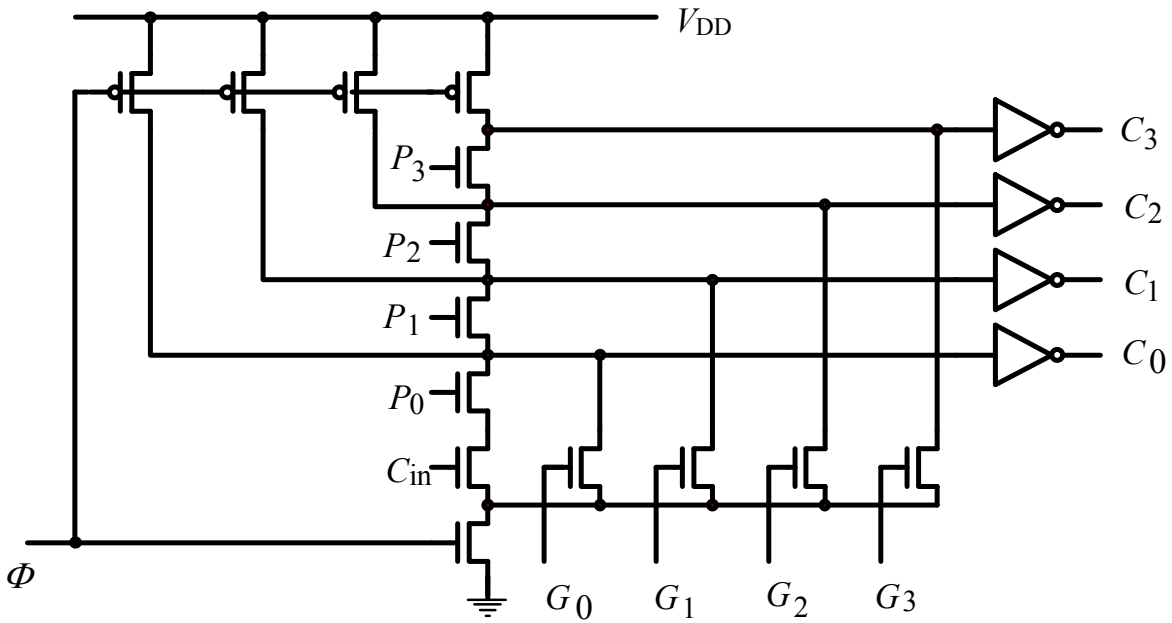
- 二进制加法
- 加法器结构设计
- 加法器电路设计

一种静态CMOS进位链电路



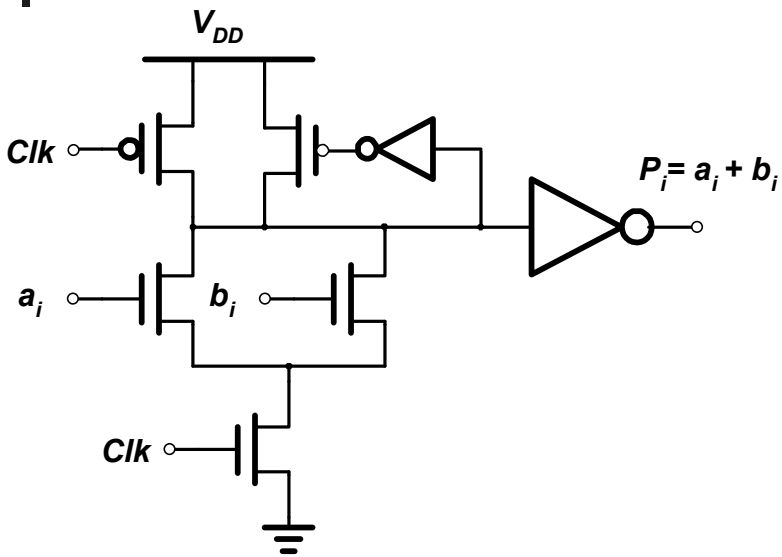
$$C_3 = G_3 + P_3C_2$$
$$= G_3 + P_3G_2 +$$
$$P_3P_2G_1 + P_3P_2P_1G_0$$
$$+P_3P_2P_1P_0C_{in}$$

进位链：多输出多米诺

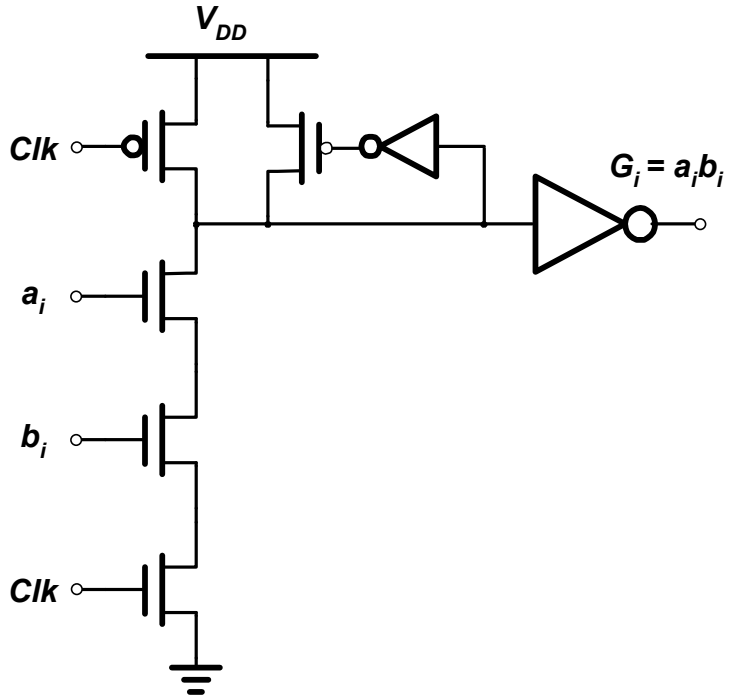


$$C_3 = G_3 + P_3 C_2$$
$$= G_3 + P_3 G_2 +$$
$$P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$
$$+ P_3 P_2 P_1 P_0 C_{in}$$

Example: Domino Adder



Propagate

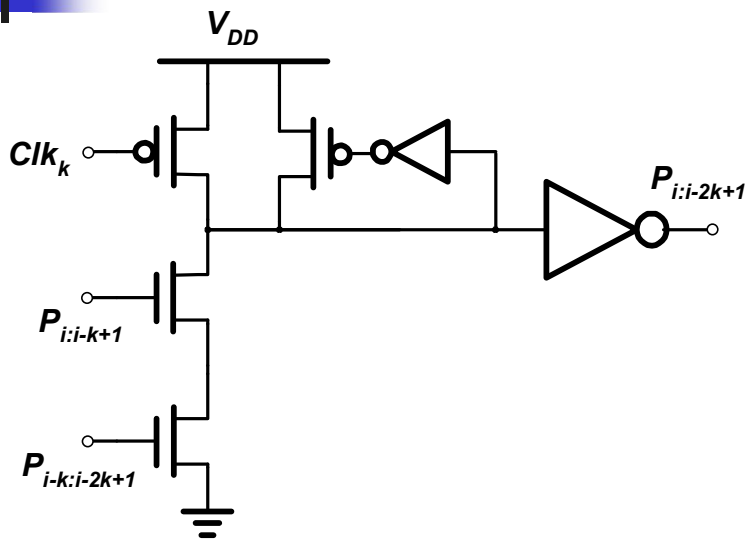


Generate

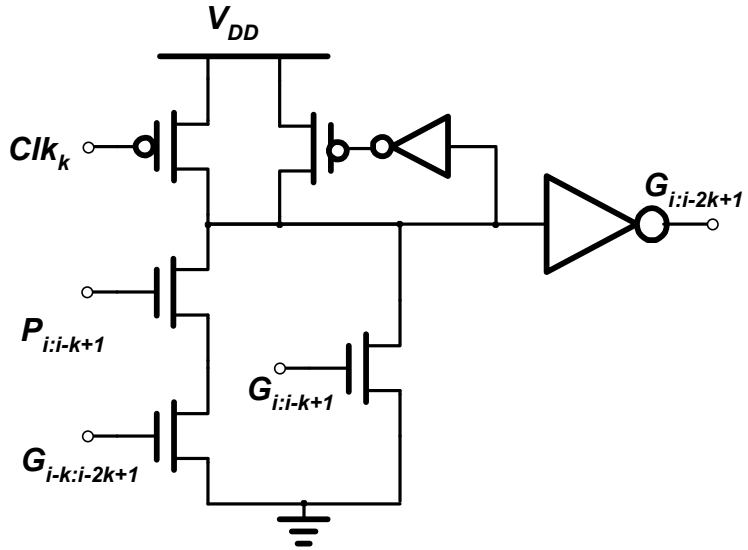
Generate (G) = AB

Propagate (P) = A + B

Example: Domino Adder



Propagate



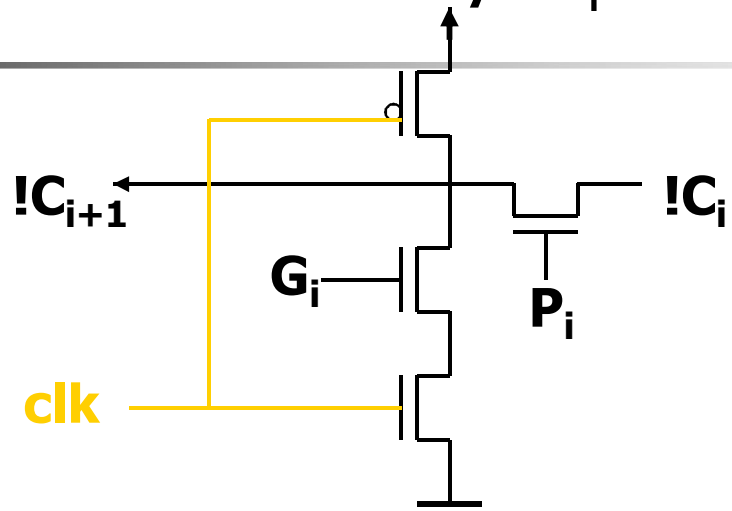
Generate

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

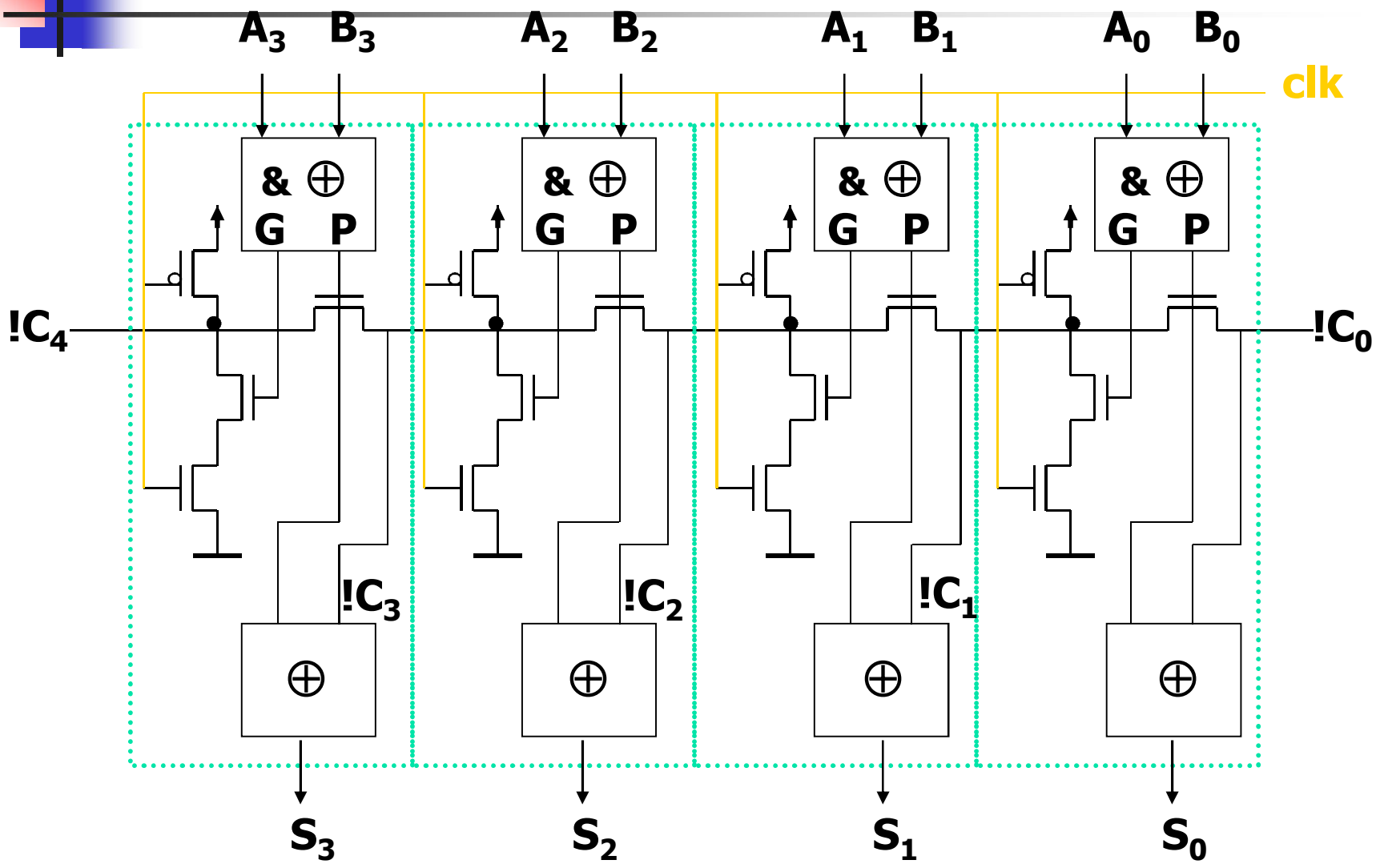
Manchester 进位链

- Switches controlled by G_i and P_i

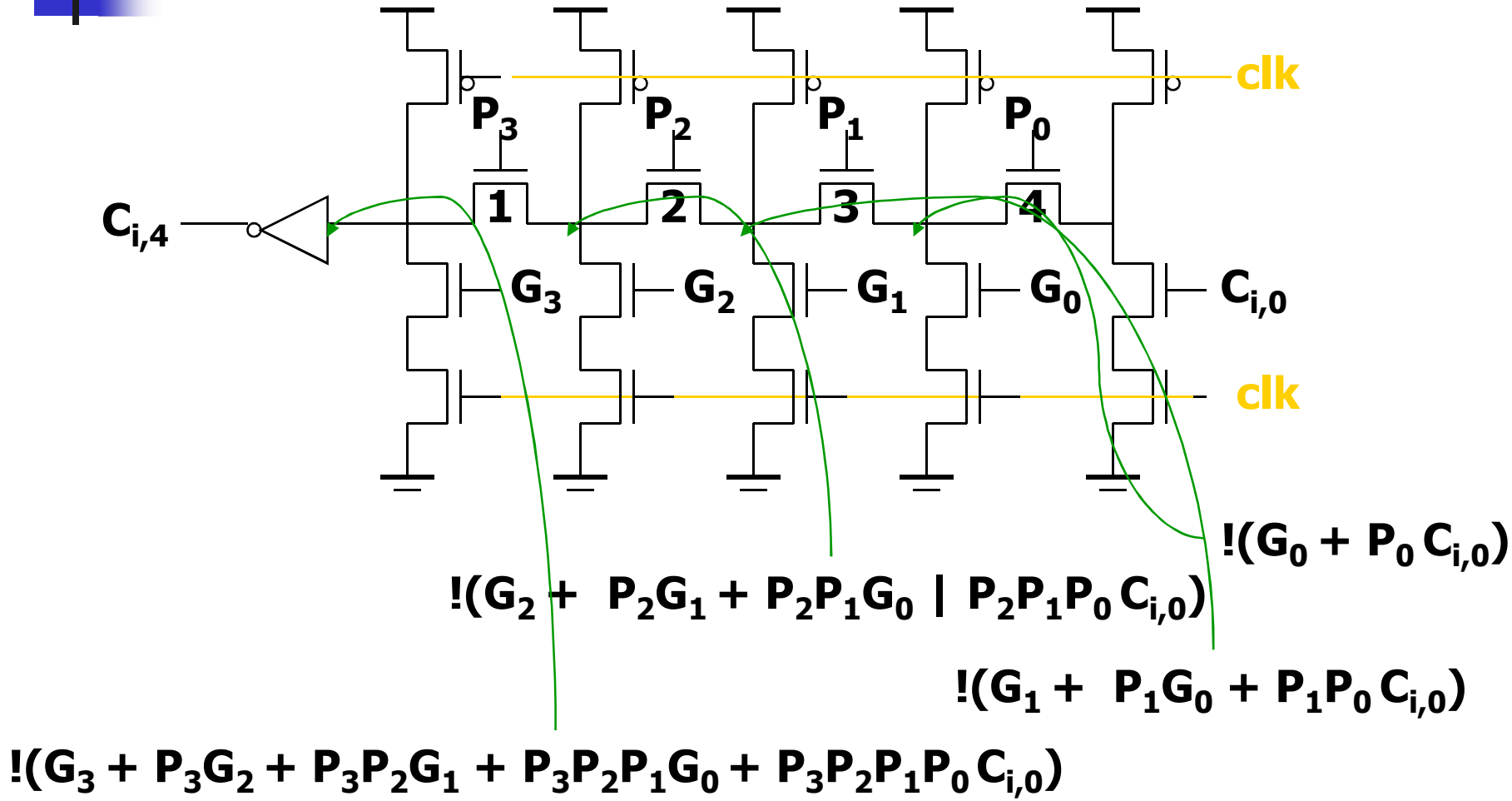


- Total delay of
  - time to form the switch control signals G_i and P_i
  - setup time for the switches
  - signal propagation delay through N switches in the worst case

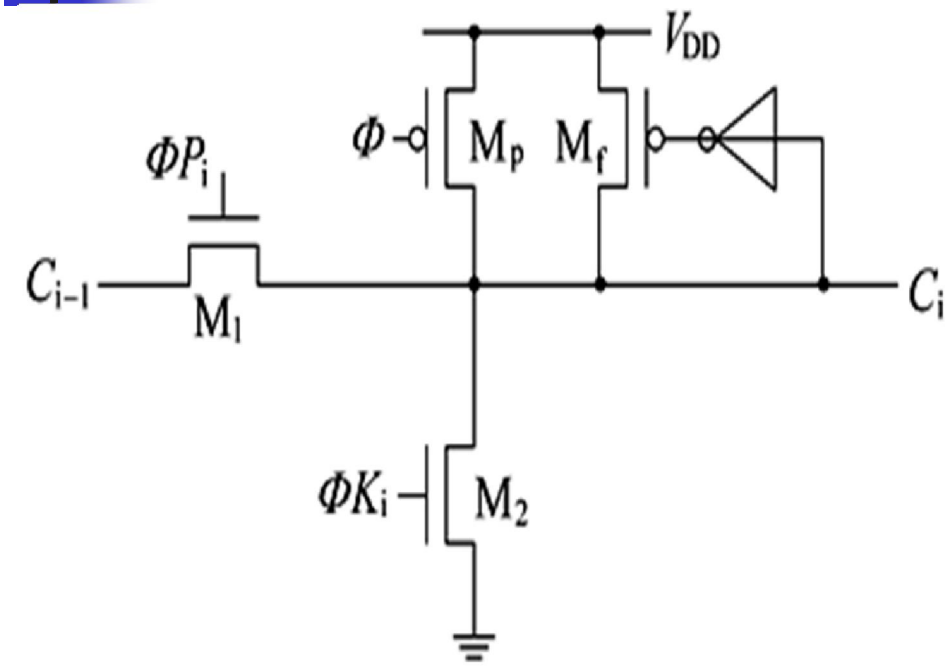
4-bit Sliced MCC Adder



Domino Manchester Carry Chain Circuit



曼彻斯特进位链



Φ=0 Mp导通，Ci预充到高电平

Φ=1 Mp截止

若Pi=1，则M1导通，Ci=Ci-1

若Ki=1，则M2导通，Ci=0

若Pi=0，Ki=0，则Ci依靠反馈管Mf维持高电平

$$K_i = \overline{A}_i \overline{B}_i$$

进位消除函数

Manchester Carry Chain

$$C_3 = G_3 + P_3 C_2$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

- Implement P with pass-transistors
- Implement G with pull-up, kill (delete) with pull-down
- Use dynamic logic to reduce the complexity and speed up